# A Parallel CORBA Component Model

Christian Pérez, Thierry Priol, André Ribes

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *A Parallel CORBA Component Model*

Christian Pérez, Thierry Priol, André Ribes

**N°4552**

Septembre 2002

THÈME 1


ISSN 0249-6399 ISRN INRIA/RR--4552--FR+ENG

# A Parallel CORBA Component Model

Christian Pérez, Thierry Priol, André Ribes

Thème 1 — Réseaux et systèmes
Projet PARIS

Rapport de recherche n° 4552 — Septembre 2002 — 15 pages

**Abstract:** The fast growth of high bandwidth wide area networks has allowed the building of grids, which are constituted of PC clusters and/or parallel machines. Grids enable the design on new numerical simulation applications. For example, it is now feasible to couple several scientific codes to obtain a multi-physic application. In order to handle the complexity of such applications, software component technology appears very appealing. However, most current software component models do not provide any support to transparently and efficiently embed parallel codes into components. This paper describes a first study of GridCCM, an extension to the CORBA Component Model to support parallel components. The feasibility of the model was evaluate thanks to the implementation of the model above two CCM prototypes. Preliminary performance results show that bandwidth is efficiently aggregated.

**Key-words:** parallel component, CCM, CORBA, SPMD code, code coupling, grid

*(Résumé : tsvp)*

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

# Un modèle de composant parallèle pour CORBA

**Résumé :** L'émergence de nouvelles technologies réseaux longue distance permet d'envisager la construction de grilles de calcul à partir de plusieurs super-calculateurs dispersés géographiquement. Ce nouveau type d'infrastructure de calcul permet de concevoir de nouvelles applications dans le domaine de la simulation numérique. Il s'agit notamment de coupler plusieurs codes de calcul scientifique pour réaliser des simulations multi-physiques. Afin de maîtriser la complexité de réalisation de telles applications, une approche par composants logiciels apparaît comme une solution prometteuse. Cependant, les modèles de composants actuels ne proposent pas de support pour encapsuler efficacement des codes parallèles. Ce papier décrit une première étude de GridCCM, une extension du modèle de composants de CORBA (CCM) supportant les composants parallèles. Un prototype, réalisé avec deux implémentations de CCM, a permis de valider la faisabilité de l'approche.

**Mots-clé :** Composant parallèle, CCM, CORBA, code SPMD, Couplage, grille de calcul

# 1 Introduction

[1] The fast growth of high bandwidth wide area networks (WAN) allows to build computing infrastructures, known as computational grids [10]. Such infrastructures, like the TeraGrid project [3], allow the interconnection of PC clusters and/or parallel machines with high bandwidth WAN. For example, the bandwidth of the French VTHD WAN [4] is 2.5 Gb/s while TeraGrid targets 40 Gb/s bandwidth.

Thanks to the performance of grid infrastructure, new kinds of applications are enabled in the scientific numerical simulation field. In particular, it is now feasible to couple scientific codes, each code simulating a particular aspect of a physical phenomenon. For example, it is possible to perform more realistic simulation of the behavior of a satellite by incorporating all aspects of a simulation: dynamic, thermal, optic and structural mechanic. Each of these aspects is simulated by a dedicated code, which is usually a parallel code, and is executed on a set of nodes of the grid. Because of the complexity of the phenomena, these codes are developed by independent specialist teams. So, one may expect to have to couple codes written in different languages (C, C++, FORTRAN, etc) using different communication paradigms (MPI, PVM, etc).

The evolution of scientific computing requires a programming model including technologies coming from both parallel and distributed computing. Parallelism is needed to efficiently exploit PC clusters or parallel machines. Distributed computing technologies are needed to control the execution and to handle communications between codes running in heterogeneous grids.

The complexity of such applications is very high with respect to design issue but also with respect to deployment issue. So, it appears that it is required to consider adequate programming models. Software component [23] appears as an appealing solution because a code coupling application can be seen as an assembly of components; each component embeds a simulation code. However, most software component models such as COM+ [20], Enterprise Java Bean [22] or CORBA Component Model [16] only support sequential component. With these models, a component is associated to a unique address space: the address space of the process where the component is created. Communication between components consists in transferring data from one address space to another one using communication mechanism such as port.

The embedding of parallel codes in sequential components raises some problems. Usually, parallel codes are executed by processes. Each process owns its private address space and uses a mechanism like MPI to exchange data with other processes (SPMD model). A first solution consists in allowing only one process to handle the component port to talk with others components. This solution leads, first, to a bottleneck in the communication between two parallel components and, second, it leads to a modification of the parallel code to introduce this master/slave pattern: the node handling the port is a kind of master, the others nodes are the slaves. A second solution would be to require that all communications

[1]This work was supported by the Incentive Concerted Action "GRID" (ACI GRID) of the French Ministry of Research.

have to be done through the software component model. This does not seem to be a good solution: first, it is a huge work to modify existing codes. Second, parallel oriented communication paradigm like MPI are much more tailored to parallelism while component communication paradigm is oriented toward distributed computing.

A better solution appears to let parallel components choose their communication paradigm and to allow all processes belonging to a parallel component to participate to inter-parallel component communications. Hence, a data transfer from the address spaces of the source parallel component to the address spaces of the destination component can generate several communication flows. It should include a data redistribution as the source and the destination data distribution may be different.

The only specification that we are aware with respect to high-performance components is the work done by the CCA Forum [6] (Common Component Architecture). The CCA Forum objectives are *"to define a minimal set of standard interfaces that a high-performance component framework has to provide to components, and can expect from them, in order to allow disparate components to be composed together to build a running application"*. The model, currently being developed, does not intentionally contain an accurate definition of a CCA component. It only defines a set of APIs. It is a low level mechanism: only point to point communications are defined. There is also a notion of collective ports, that allows a component to broadcast data to all the components connected to this port. While CCA's goal is to define a model specifically targets high performance computing, we aim to adapt existing standards to high performance computing.

There are several works about parallel objects like PARDIS [13] or PaCO [21]. OMG has started a normalization procedure to add data parallel features to CORBA. The adopted specification [17] requires modifications to the ORB (the CORBA core). Another work, PaCO++ [8], is an evolution of PaCO that targets efficient and portable parallel CORBA objects.

The contribution of this paper is to study the feasibility of defining and implementing parallel components within the CORBA Component Model (CCM). We choose to work with CCM because it inherently supports the heterogeneity of processors, operating system and languages; it is an open standard and there are several implementations being available with an Open Source license. Moreover, the model provides a deployment model. CORBA seems to have some limitation but it appears possible to overcome most of them. For example, we have shown that high performance can be achieved [9]. The problems related with IDL can be solved by defining and using domain specific types or valuetypes to handle complex number or graph types.

Our objective is to obtain both parallelism transparency and high performance. Transparency is required because a parallel component needs to look like a standard component at the assembly phase. The effective communications between parallel components needs to be hidden to the application designer. To obtain high performance, we propose to apply a technique that allows all processes of a parallel component to be involved in inter-component communications. Thus, inter-parallel component communications are able to remain efficient when increasing the number of node of a parallel component.

The remaining of this paper is divided as follows. Section 2 presents a brief discussion about objects and components. The CORBA component model is presented in Section 3. GridCCM, our parallel extension to the CORBA component model, is introduced in Section 4. Some preliminary experiment results are reported in Section 5. Section 6 concludes the paper.

# 2 From objects to components

Object-oriented programming has provided substantial advantages over structured programming. Recently, software component technology is expected to bring a similar evolution to software technology. While object-oriented programming targets application design, component software technology emphasizes component composition and deployment. Before introducing the software component technology, the object-oriented programming model is briefly reviewed.

## 2.1 Object-oriented programming

Object is a powerful abstraction mechanism. It has demonstrated its benefits, in particular in application design, in a very large number of applications.

However, objects have failed some of their objectives. Code reuse and maintenance are not satisfactory mainly because object dependences are not very explicit. For example, it is very difficult to find object dependences in a large application involving hundreds of objects using inheritance and delegation. Experience has shown that it is better to use an approach only based on delegation [11] that allows objects to be "composed".

For distributed applications, objects do not intrinsically provide any support for deployment. For example, neither JAVA RMI [12] nor CORBA 2 provide a way to remotely install, create or update an object.

Despite its benefits, object oriented programming lacks some important features for distributed applications. Software components aim at providing them.

## 2.2 Software component

Software component technology [23] has been emerging for some years [7] even though its underlying intuition is not very recent [15]. Among all the definition of software components, here is Szyperski's one [23]: *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*

First, the main component operation is the composition with other components. This composition is done through well-defined interfaces: components need to agree on the contract related to their connected interfaces. This agreement brings a lot of advantages: it explicits the abstraction of the service. In particular, interfaces are strongly typed so that checks such as type conformance can be performed at connection time.

Second, a component inherently embeds deployment capabilities. All the information like the binary code (i.e. the code of the component), the dependencies in term of processors, operating systems and libraries are embedded into the component. A component may also embed binary codes for different processors or operating systems. The right version of the binary is automatically selected by the deployment tool. So, deployment in a heterogeneous environment is easier.

Building an application based on components emphasizes programming by *assembly*, i.e. manufacturing, rather than by *development*. Some goals are to focus expertise on domain problems, to improve software quality and to decrease the time to market thanks to reuse of existing code.

Components exhibit advantages over objects. Applications are naturally more modular as each component represents a functionality. Code reuse and code maintainability are ease as component are well-defined and independent. Last, components provide mechanisms to be deployed and to be connected in a distributed infrastructure. Thus, they appear very well suited for grid computing.

# 3 CORBA Component Model

The CORBA Component Model [16] (*CCM*) is going to be added to the next CORBA [18] version (version 3). The CCM specifications allow the deployment of components into a distributed environment, that is to say that an application can deploy interconnected components on different servers.

A CORBA component is represented by a set of ports described in IDL3, an extension of CORBA's IDL2 language. There are five kinds of ports as shown in Figure 1. Facets are distinct named interfaces provided by the component for client interaction. Receptacles are named connection points that describe the component's ability to use a reference supplied by some external agent. Event sources are named connection points that emit events of a specified type to one or more interested event customers, or to an event channel. Event sinks are named connection points into which events of a specified type may be pushed. Attributes are named values exposed through accessor and mutator operations. Attributes are primarily intended to be used for component configuration, although they may be used in a variety of other ways.

Each component's life cycle is managed by an entity named *home*. To create a component, a client calls a **create** method of the component home interface.

Figure 2 shows how a component **ServerComp** is connected to component **ClientComp** through the facet **FacetExample**. First, a reference is obtained from the facet. Second, this reference is given to a receptacle. This connection is dynamically done by a third party, usually the deployment tool. Consequently, connexions can be changed during component's life.

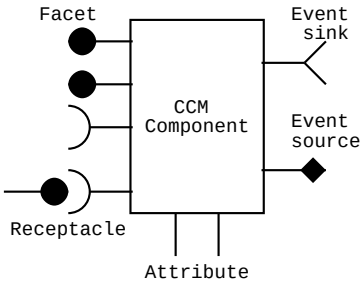


Figure 1: A CCM component

```
foo ref =
   ServerComp->provide_FacetExample();
ClientComp->connect_FacetExample(ref);
```

Figure 2: Example of code to connect two components.

# 4 A Parallel CORBA Component Model: GridCCM

The CORBA component model is an interesting component model mainly because its deployment model. However, it provide no efficient support to embed parallel codes into components. This section presents GridCCM, a parallel extension to the CORBA Component Model (CCM). A GridCCM's goal is to study the concept of parallel component.

## 4.1 Overview

Our objective is to encapsulate parallel codes into CORBA components with as few modifications to parallel codes as possible. Similarly, we target to extend CCM but not to introduce deep modifications in the model. That's why, we don't allow ourselves to modify the CORBA's Interface Definition Language (IDL).

We currently restrict ourselves to embed SPMD (*Single Program Multiple Data*) codes into a parallel component. In a SPMD code, each process executes the same program but on different data. This choice stems from two considerations. First, many parallel codes are indeed SPMD. Second, SPMD codes bring a manageable execution model.

Figure 3 shows our vision of a parallel component in the CORBA framework. The SPMD code continues to be able to use MPI for its inter-process communications but it uses CORBA to communicate with other components. In order to avoid bottlenecks, all processes of a parallel component participate to inter-component communications. This scheme has been successfully demonstrated with parallel CORBA object [8]: an aggregated bandwidth of 103 MB/s (820 Mb/s) has been obtained on VTHD [4], a French high-bandwidth WAN, between two 11-nodes CORBA parallel objects.

GridCCM's objective is to extend CCM to allow an efficient encapsulation of SPMD codes. For example, GridCCM has to be able to aggregate bandwidth when two parallel components exchange data. As data may need to be redistributed during communications, GridCCM model has to support it as transparently as possible. The client should only have to describe how its data are locally distributed and the data should be automatically redistribute accordingly to the server's preferences. A GridCCM component should appear

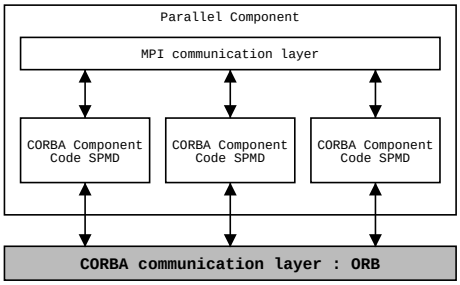


Figure 3: Parallel component concept

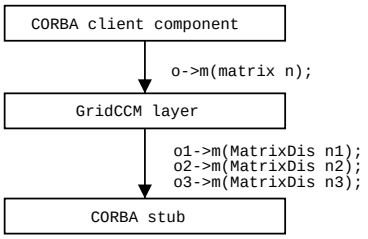


Figure 4: The user invokes a method on a remote component. The GridCCM layer actually invokes the distributed version of this method.

as close as possible to a *sequential* component. For example, a sequential component should connect itself to a parallel component without noticing it is a *parallel* component.

To achieve these objectives, GridCCM introduces the notion of parallel component. Our definition of a parallel component is:*a parallel component is a collection of identical sequential components. It executes in parallel all or some parts of its services.*

The designer of a parallel component needs to describe the parallelism of the component in an auxiliary XML file. This file contains a description of the parallel methods of the component, the distributed arguments of these methods and the expected distribution of the arguments. An example is given in section 4.4.

## 4.2 Introducing parallelism support into CCM

To introduce parallelism support, like data redistribution, without requiring modifications to the ORB, we choose to introduce a new software layer between client's code and stub's code as illustrated in Figure 4. This scheme has been successfully used with PaCO++ [8] for a similar problem: the management of parallel CORBA objects.

The role of the GridCCM layer is to allow a transparent management of the parallelism. A call to a parallel method of a parallel component is intercepted by this new layer. The layer sends the distributed data from the client nodes to the server nodes. The data redistribution can actually be performed on the client side, on the server side or during the communication between the client and the server. The decision depends on several constraints like feasibility (mainly memory requirements) and efficiency (client network performance versus server network performance). Another goal is to manage parallel exceptions [14].

The parallel management layer is generated by a compiler specific to GridCCM, as illustrated in Figure 5. This compiler uses two files: an IDL description of the component and a XML description of the component parallelism. In order to have a transparent layer, a new IDL description is generated during the generation of the component. GridCCM layer internally uses an interface derived from the original interface. The new IDL interface is

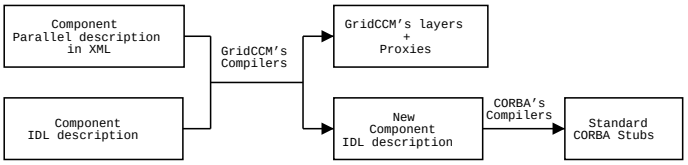


Figure 5: The different compilation phase to generate a parallel component

the interface that is remotely invokes on the server side. The original IDL interface is used between the user code and the GridCCM layer on the client and the server sides.

In the new IDL interface, the user arguments described as distributed have been replaced by their equivalent distributed data types. Because of this transformation, there are some constraints about the types that can be distributed. The current implementation requires the user type to be an IDL `sequence` type, that is to say a 1D array. So, one dimension distribution can automatically be applied. This scheme can easily be extended for multidimensional arrays: a 2D array can be mapped to a sequence of sequences. It is worthwhile to note that IDL type do not allow a direct mapping of "scientific" types like multidimensional arrays or complex number. However, CORBA is just a communication technology. Higher level environments, like code coupling environments such as HLA [5] or MpCCI [2], should be able to overcome the CORBA type system limitation as described in the introduction.

## 4.3 Home and Component proxies

One of our objectives is to allow a parallel component to be seen as a sequential component. So, the nodes of a parallel component and the nodes of the component homes are not directly exposed to other components. We introduced two entities, named the *HomeManager* and the *ComponentManager*. They are proxies respectively for the nodes homes and for the parallel component's nodes.

An application that needs to create a parallel component interacts in the CCM standard way with the *HomeManager* instead of using each nodes home. The *HomeManager* is configured during the deployment phase. The references to all the nodes homes are given through a specific interface to the *HomeManager*. Then, the *HomeManager* calls the homes on all the nodes.

Similarly, when a client gets a reference to a parallel component, it actually has a reference to the *ComponentManager*. When two parallel components are connected, the *ComponentManager*s transparently exchange information about the parallel components so as to configure GridCCM's layers.

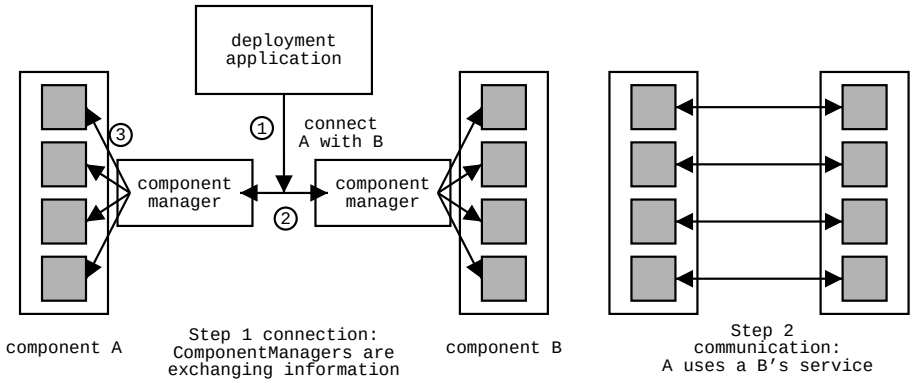


Figure 6: Connection between two parallel components

Figure 6 shows an example of connection between two parallel components: A and B. First, the deployment tool connects A with B using the standard CCM protocol (see Figure 2). Second, A's *ComponentManager* asks B's *ComponentManager* for information. For example, component B gives the references of all B's nodes to component A. Third, A's *ComponentManager* configures the layers of all A's nodes. Fourth, when component A performs a call to a B's service, all A's nodes may participate to the communication with B's nodes.

## 4.4 Example

This example illustrates the definition and the use of a parallel component.

A component, named *A*, provides a port of name *myPort* which contains an operation that computes the average of a vector. Figure 7 shows the IDL description of the component and the IDL description of the *Average* interface. The facet and the interface are described without CCM's IDL modifications.

The component implementer needs to write an XML file to describe the parallelism of component *A*. This file, shown in figure 8, is not presented in XML for the sake of clarity. An important remark is that this file is not seen by the client. In this example, the operation *compute* is declared parallel and its first argument is block distributed.

Standard clients normally use the *myPort* facet. A parallel client has to specify how the data to be sent are locally distributed. Figure 9 shows an example of the API. Its specification is not yet stable mainly because we work at allowing external redistribution library [1] to be plugged. In the example of Figure 9, the client gets a reference of the facet *myPort*. Then, it configures its local GridCCM layer with the method *init_ compute* before invoking the *compute* operation.

```
// Average definition
typedef sequence<long> Vector;
interface Average {
  long compute(in Vector v);
};
// ExParComponent definition
component A {
  provides Average myPort;
};
```

Figure 7: A component IDL definition

```
// Parallelism definition
Component: A
Port     : myPort
Port Type: Average
Operation: compute
Argument1: bloc
Result   : noReduction
```

Figure 8: Parallelism description of the facet

```
 // Get a reference to the facet myPort
Average avg = aComponent.get_connection_myPort();
 // Configure it with a GridCCM API
avg.init_compute('bloc',0); // 1st arg is bloc distributed
 // "Standard" CCM call
avg.compute(MyMatrixPart);
```

Figure 9: A client initializes and uses a parallel operation

# 5 Preliminary experiments

This section presents some preliminary experiments that evaluate the feasibility and the scalability of GridCCM. They also show that the model is generic with respect to two different CCM implementations. First, the experimental protocol is presented and, second, some performance measurements are reported.

## 5.1 Experimental protocol

The experimental setup, shown in Figure 10, contains two parallel components (*Customer* and *Server*), a GridCCM-aware sequential component and a standard sequential CORBA client. First, the standard CORBA client creates a vector and sends it to the sequential component. Second, the vector is sent to *Customer* with an automatic distribution in *Customer*'s nodes according to a bloc distribution. Third, *Customer* invokes a *Server*'s service which takes the distributed vector as an in-out argument. The *Server* method implementation contains only a MPI barrier. Then, the vector is sent back to *Customer*. In this example, the two parallel components use MPI. *Customer* uses MPI for barriers and reductions. *Server* only uses MPI for barriers.

There is no modification in the sequential component code to call a *Customer*'s parallel service. The parallel service is transparent for the sequential code. With regard to the connection, the deployment application complies with the CCM specification. The *Compo-*

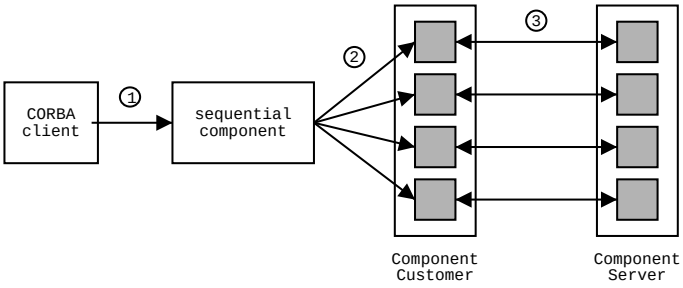


Figure 10: Experimental protocol

| Number of nodes of parallel components | Aggregate bandwidth in MB/s | |
|---|---|---|
| | Java compiled (JIT) | C++ |
| 1 | 8.3 | 9.8 |
| 2 | 16.6 | 19.6 |
| 4 | 33.2 | 39.2 |
| 8 | 66.4 | 78.4 |

Figure 11: Bandwidth between component *Customer* and component *Server*. The two parallel components have with the same cluster size.

*nentManager*s perform their role correctly since the parallel component nodes are connected without user intervention.

## 5.2 Performance measurements

We have implemented a preliminary prototype of GridCCM on top of two CCM implementations: OpenCCM [24] and MicoCCM [19]. Our first prototype has been derived from OpenCCM [24]. OpenCCM is made by the research laboratory LIFL (*Laboratoire Infomatique Fondamentale de Lille*). It is written in Java. The second prototype has been derived from MicoCCM [19]. MicoCCM is an *OpenSource* implementation based on the Mico ORB. It is written in C++.

The test platform is a cluster of 16 dual-pentium III 1 Ghz with 512 MB of memory interconnected with a Fast Ethernet network and a gigabit switch. For OpenCCM, the Java Runtime Environment is the SUN JDK 1.3 and ORBacus 4.0.5 is the used ORB.

The measures have been made in the first parallel component (*Customer*). After a MPI barrier to synchronize all *Customer*'s nodes, the start time is measured. *Customer* performs 1000 calls to *Server*. Then, the end time is taken.

The aggregated bandwidth between the two parallel components is shown in Figure 11 for different sizes of the parallel components. First, bandwidth is efficiently aggregated. Second, as expected C++ implementation is more efficient than the Java one. Last, the latency for the 8-node to 8-node case is 215 $\mu$s for MicoCCM. This is the latency observed for plain MicoCCM over a Fast Ethernet network. The GridCCM layer, without data redistribution, does not add a significant overhead.

In conclusion, these preliminary experiments show that parallel components can be achieved without exhibiting overhead on classical networks. In the future, we plan to evaluate more accurately GridCCM overhead using PadicoTM [9] to have a high performance CORBA environment. With PadicoTM, a point-to-point latency of 20 $\mu$s and a point-to-point bandwidth of 240 MB/s have been measured on a Myrinet-2000 network. On such network, MPI has a latency of 11 $\mu$s and a bandwidth of 240 MB/s.

# 6 Conclusion

Computing grids allow new kinds of application to be developed. For example, code coupling applications can benefit a lot from grids because grids provide very huge computing, networking and storage resources. However, the distributed and heterogeneous nature of grids raises many problems to application designer. Software component technology appears to be a very promising technology to handle such problems. However, software component models do not offer a transparent and efficient support to embed parallel codes into components.

The first contribution of this paper is to describe a first study of the extension of the CORBA Component Model in order to introduce parallel components. Another contribution is to have successfully integrated the proposed model into two existing CCM prototypes. With both prototypes, the preliminary performance results have shown that the model is able to aggregated bandwidth without introducing noticeable overhead. However, more experiments need to be done.

The next step of our work is to finalize the GridCCM model, in particular with respect to parallel component exceptions. Regarding to security issue, a parallel component should behave as a standard CORBA component. Supporting fault tolerance feature is a more complex issue that needs to be further investigated. Moreover, the prototype needs to be finalized. In particular, the current GridCCM layer is mainly hand-written. A dedicated compiler will soon be operational. Last, we plan to test the model with real applications. One of them will be an EADS code coupling application which involves five MPI codes.

# References

[1] The darpa data reorganization effort. http://www.data-re.org.

[2] Mpcci - mesh-based parallel code coupling interface. http://www.mpcci.org.

[3] The teragrid project. http://www.teragrid.org.

[4] The VTHD project. http://www.vthd.org.

[5] Ieee standard for modeling and simulation (m&s) high level architecture (hla)—federate interface specification, 2000.

[6] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proceeding of the 8th IEEE International Symposium on High Performance Distributed Computation*, Août 1999.

[7] L. Barroca, J. Hall, and P. Hall. *Software Architectures: Advances and Applications*, chapter An Introduction and History of Software Architectures, Components, and Reuse. Springer Verlag, 1999.

[8] A. Denis, C. Pérez, and T. Priol. Portable parallel corba objects: an approach to combine parallel and distributed programming for grid computing. In *Proc. of the 7th Intl. Euro-Par'01 conf.*, pages 835–844, Manchester, UK, 2001. Springer.

[9] A. Denis, C. Pérez, and T. Priol. Towards high performance CORBA and MPI middleware for grid computing. In *Proc of the 2nd International Workshop on Grid Computing*, Denver, Colorado, USA, November 2001.

[10] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc, 1998.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, 1995.

[12] William Grosso. *Java RMI*. O'Reilly & Associates, 2001.

[13] K. Keahey and D. Gannon. PARDIS: A Parallel Approach to CORBA. In *Supercomputing'97*. ACM/IEEE, November 1997.

[14] Hanna Klaudel and Franck Pommereau. A concurrent semantics of static exceptions in a parallel programming language. In J.-M. Colom and M. Koutny, editors, *Applications and theory of Petri nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 204–223. Springer, 2001.

[15] M. D. McIlroy. Mass Produced Software Components. In P. Naur and B. Randell, editors, *Software Engineering*, pages 138–155, Brussels, 1969. Scientific Affairs Division, NATO.

[16] OMG. Corba 3.0 new components chapters, nov 2001. Document ptc/2001-11-03.

[17] OMG. Data parallel CORBA. Technical report, 2001. Document orbos/01-10-19.

[18] OMG. The Common Object Request Broker: Architecture and Specification (Revision 2.5). OMG Document formal/01-09-34, September 2001.

[19] Frank Pilhofer. The mico corba component project. http://www.fpx.de/MicoCCM.

[20] David S. Platt. *Understanding COM+*. Microsoft Press, 1999.

[21] C. René and T. Priol. MPI code encapsulating using parallel CORBA object. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, pages 3–10, August 1999.

[22] E. Specification and S. June. Enterprise JavaBeans Specification, 1999.

[23] C. Szyperski. *Component Software - Beyond Object-Oriented Programming.* Addison-Wesley / ACM Press, 1998.

[24] M. Vadet, P. Merle, R. Marvie, and J.-M. Geib. The OpenCCM platform. http://corbaweb.lifl.fr/OpenCCM/.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399